www.startimes2.com

دروس في الـ OpenGL

الدرس 5: التحكم في شخصيات Quake 2’s MD2

khatibe_30@hotmail.fr



منتدى لغات البرمجة
www.startimes2.com

2010

الملفات ذات اللاحقة md2 هي ملفات تحتوي على موديلات 3D وهي مصممة خصيصا للعبة المشهورة Quake 2, هي فعلا قديمة (1997) ولكنها بسيطة و سهلة الفهم للمبتدئين, كما قلنا سابقا, يحتوي الملف من نوع MD2 على شخصية أو أي موديل خاص باللعبة, يمكنك إنشاء شخصيات ثلاثية الأبعاد و حفظها في ملف md2 باستعمال أحد برامج التصميم مثل MilkShape 3D.

قبل الخوض في ملفات md2 افتح الفيجوال ستديو و أنشئ مشروع جديد من نوع console و سمه Lesson5 و أضف إليه ملف جديد باسم main.cpp و اكتب في الملف main.cpp نفس الكود الذي رأيناه في الدرس السابق و الذي ينشئ نافذة OpenGL و أرضية مع أربعة جدران و كذلك اكتب الكود الذي يسمح لنا بالتجول داخل العالم الثلاثي الأبعاد بحرية, و لكن سنجري تغييرا على طريقة تحميل الإكساءات من الصور, لذلك أضف إلى المشروع ملفين أحدهما textures.h و الآخر textures.cpp والواضح من أسمائهما أننا سنكتب بهما الدوال المسؤولة عن الإكساء و لكن هذه المرة سنحمل الإكساء من ملفات bmp و tga و pcx, أكتب في الملف textures.h هذا الكود:

```
#pragma once
#ifndef         __TEXTURES_H_
#define         __TEXTURES_H_

///////////////////              BMP              //////////////////
// magic number "BM"
#define               BITMAP_ID            ('B' + ('M'<<8))
// header byte type for RLE
#define               RLE_COMMAND          0
#define               RLE_ENDOFLINE        0
#define               RLE_ENDOFBITMAP      1
#define               RLE_DELTA            2

#define               BI_OS2                     -1

int LoadFileBMP(char *filename, unsigned char **pixels, int *width, int
*height, bool flipvert);
///////////////////              PCX              //////////////////
#pragma warning( disable : 4103 ) // used #pragma pack to change alignment
// --------------------------------------------
// PCXHEADER - pcx header structure.
// --------------------------------------------
#pragma pack(1)
typedef struct tagPCXHEADER
{
      unsigned char     manufacturer;          // manufacturer
      unsigned char     version;               // version
      unsigned char     encoding;              // encoding type
      unsigned char     bitsPerPixel;          // number of bits per pixel

      unsigned short    x, y;                  // ...
      unsigned short    width, height;         // dimensions
      unsigned short    horzRes, vertRes; // horisontal and vertical screen
resolutions
      unsigned char     *palette;              // color palette
      unsigned char     reserved;              // reserved
      unsigned char     numColorPlanes;        // number of planes
      unsigned short    bytesPerScanLine; // byte per line
      unsigned short    paletteType;           // palette type
      unsigned short    horzSize, vertSize;    // ...
      unsigned char     padding[54];           // ...

} PCXHEADER, *PPCXHEADER;
```

```
#pragma pack(4)
int LoadFilePCX( const char *filename, unsigned char **pixels, int *width, int
*height, bool flipvert );
///////////////////            TGA              //////////////////
#pragma warning( disable : 4103 ) // used #pragma pack to change alignment
// -----------------------------------------------
// TGAHEADER - targa header.
// -----------------------------------------------
#pragma pack(1)
typedef struct tagTGAHEADER
{
	unsigned char	id_lenght;		// size of the structure
	unsigned char	color_map_type;		// must be equal to 0
	unsigned char	image_type;		// image compression type

	short int	cm_first_entry;		// colormap_origin (toujours 0)
	short int	cm_length;		// colormap_length (toujours 0)
	unsigned char	cm_size;		// colormap_size (toujours 0)

	short int	is_xorigin;		// lower left X coordinate
(always 0)
	short int	is_yorigin;		// lower left Y coordinate
(always 0)

	short int	is_width;		// image width (in pixels)
	short int	is_height;		// image height (in pixels)

	unsigned char	is_pixel_depth;		// number of bits per pixel:
16, 24, 32
	unsigned char	is_image_descriptor;// 24 bits = 0x00; 32 bits = 0x80
} TGAHEADER, *PTGAHEADER;
#pragma pack(4)
// -----------------------------------------------
// BGRAQUAD - 32 bits pixel
// -----------------------------------------------
typedef struct tagBGRAQUAD			// rgbt
{
	unsigned char	bgraBlue;		// blue
	unsigned char	bgraGreen;		// green
	unsigned char	bgraRed;		// red
	unsigned char	bgraAlpha;		// alpha

} BGRAQUAD, *PBGRAQUAD;
int LoadFileTGA( const char *filename, unsigned char **pixels, int *width, int
*height, bool flipvert );
//////////////////////////////////////////////////////////////////////
unsigned int LoadTexture(char *filename);

#endif //__TEXTURES_H_
```

قمنا بالتصريح عن أربع دوال:

LoadFileBMP : لقراءة بيانات صورة من نوع bmp
LoadFilePCX : لقراءة بيانات صورة من نوع pcx
LoadFileTGA : لقراءة بيانات صورة من نوع tga
LoadTexture : لتحميل الإكساء من الملف و تعيد معرف الإكساء

أما في الملف textures.cpp فنكتب كود أجسام الدوال السابقة :

```
#include "textures.h"
#include <stdlib.h>
#include <windows.h>
#include <GL\glut.h>
#include <stdio.h>

int LoadFileBMP(char *filename, unsigned char **pixels, int *width, int
*height, bool flipvert)
{
FILE*        file;              // file stream
BITMAPFILEHEADER  *bmfh;                    // bitmap file header
BITMAPINFOHEADER  *bmih;                    // bitmap info header (windows)
BITMAPCOREHEADER  *bmch;                    // bitmap core header (os/2)
RGBTRIPLE                 *os2_palette;     // pointer to the color palette os/2
RGBQUAD                         *win_palette;     // pointer to the color palette
windows
char                      *buffer;          // buffer storing the entire file
unsigned char             *ptr;             // pointer to pixels data
int                             bitCount;         // number of bits per pixel
int                             compression;      // compression type (rgb/rle)
int                             row, col, i;      // temporary variables
int                             w, h;             // width, height
/////////////////////////////////////////////////////
// read the entire file in the buffer
file = fopen( filename, "rb" );
if( file == NULL )
return 0;
fseek(file, 0, SEEK_END);
long flen = ftell(file);
fseek(file, 0, SEEK_SET);
buffer = new char[ flen + 1 ];
fread(buffer, flen, 1, file);
char *pBuff = buffer;
fclose(file);
/////////////////////////////////////////////////////
// read the header
bmfh = (BITMAPFILEHEADER *)pBuff;
pBuff += sizeof( BITMAPFILEHEADER );
// verify that it's a BITMAP file
printf("%c", bmfh->bfType);
if( bmfh->bfType != BITMAP_ID )
{
delete [] buffer;
return 0;
}
bmch = (BITMAPCOREHEADER *)pBuff;
bmih = (BITMAPINFOHEADER *)pBuff;
if( (bmih->biCompression < 0) || (bmih->biCompression > 3) )
{
// OS/2 style
pBuff += sizeof( BITMAPCOREHEADER );
bitCount    = bmch->bcBitCount;
compression = BI_OS2;
w = bmch->bcWidth;
h = bmch->bcHeight;
}
else
{
```

```
// WINDOWS style
pBuff += sizeof( BITMAPINFOHEADER );
bitCount     = bmih->biBitCount;
compression = bmih->biCompression;
w = bmih->biWidth;
h = bmih->biHeight;
}
if( width )
*width       = w;
if( height )
*height      = h;
if( !pixels )
{
delete [] buffer;
return (-1);
}
/////////////////////////////////////////////////////
// read the palette
if( bitCount <= 8 )
{
// 24 and 32 bits images are not paletted
// ajust the palette pointer to the memory in the buffer
os2_palette = (RGBTRIPLE *)pBuff;
win_palette = (RGBQUAD *)pBuff;
//     [number of colors in the palette] * [size of one pixel]
pBuff += (1 << bitCount) * (bitCount >> 3) * sizeof( unsigned char );
}
/////////////////////////////////////////////////////
// allocate memory to store pixel data
*pixels = new unsigned char[ w * h * 4 ];
ptr          = &(*pixels)[0];
// move the pixel data pointer to the begening of bitmap data
pBuff = buffer + (bmfh->bfOffBits * sizeof( char ));
/////////////////////////////////////////////////////
// read pixel data following the image compression
// type and the number of bits per pixels
/////////////////////////////////////////////////////
switch( compression )
{
case BI_OS2:
case BI_RGB:
{
for( row = h - 1; row >= 0; row-- )
{
if( flipvert )
ptr = &(*pixels)[ row * w * 4 ];
switch( bitCount )
{
case 1:
{
// RGB 1 BITS
for( col = 0; col < (int)(w / 8); col++ )
{
// read the current pixel
unsigned char color = *((unsigned char *)(pBuff++));
for( i = 7; i >= 0; i--, ptr += 4 )
{
      // convert indexed pixel (1 bit) into rgba (32 bits) pixel
      int clrIdx = ((color & (1<<i)) > 0);
```

```
if( compression == BI_OS2 )
      {
            ptr[0] = os2_palette[ clrIdx ].rgbtRed;
            ptr[1] = os2_palette[ clrIdx ].rgbtGreen;
            ptr[2] = os2_palette[ clrIdx ].rgbtBlue;
            ptr[3] = 255;
      }
      else
      {
            ptr[0] = win_palette[ clrIdx ].rgbRed;
            ptr[1] = win_palette[ clrIdx ].rgbGreen;
            ptr[2] = win_palette[ clrIdx ].rgbBlue;
            ptr[3] = 255;
      }
}
}
case 4:
{
// RGB 4 BITS
for( col = 0; col < (int)(w / 2); col++, ptr += 8 )
{
// read the current pixel
unsigned char color = *((unsigned char *)(pBuff++));
// convert indexed pixel (4 bits) into rgba (32 bits) pixel
int clrIdx;
if( compression == BI_OS2 )
{
      clrIdx = (color >> 4);
      ptr[0] = os2_palette[ clrIdx ].rgbtRed;
      ptr[1] = os2_palette[ clrIdx ].rgbtGreen;
      ptr[2] = os2_palette[ clrIdx ].rgbtBlue;
      ptr[3] = 255;

      clrIdx = (color & 0x0F);
      ptr[4] = os2_palette[ clrIdx ].rgbtRed;
      ptr[5] = os2_palette[ clrIdx ].rgbtGreen;
      ptr[6] = os2_palette[ clrIdx ].rgbtBlue;
      ptr[7] = 255;
}
else
{
      clrIdx = (color >> 4);
      ptr[0] = win_palette[ clrIdx ].rgbRed;
      ptr[1] = win_palette[ clrIdx ].rgbGreen;
      ptr[2] = win_palette[ clrIdx ].rgbBlue;
      ptr[3] = 255;

      clrIdx = (color & 0x0F);
      ptr[4] = win_palette[ clrIdx ].rgbRed;
      ptr[5] = win_palette[ clrIdx ].rgbGreen;
      ptr[6] = win_palette[ clrIdx ].rgbBlue;
      ptr[7] = 255;
}
}
break;
}
case 8:
{
// RGB 8 BITS
for( col = 0; col < w; col++, ptr += 4 )
```

```
{
// read the current pixel
unsigned char color = *((unsigned char *)(pBuff++));

// convert indexed pixel (8 bits) into rgba (32 bits) pixel
if( compression == BI_OS2 )
{
      ptr[0] = os2_palette[ color ].rgbtRed;
      ptr[1] = os2_palette[ color ].rgbtGreen;
      ptr[2] = os2_palette[ color ].rgbtBlue;
      ptr[3] = 255;
}
else
{
      ptr[0] = win_palette[ color ].rgbRed;
      ptr[1] = win_palette[ color ].rgbGreen;
      ptr[2] = win_palette[ color ].rgbBlue;
      ptr[3] = 255;
}
}

break;
}

case 24:
{
// RGB 24 BITS
for( col = 0; col < w; col++, ptr += 4 )
{
// convert bgr pixel (24 bits) into rgba (32 bits) pixel
RGBTRIPLE *pix = (RGBTRIPLE *)pBuff;
pBuff += sizeof( RGBTRIPLE );

ptr[0] = pix->rgbtRed;
ptr[1] = pix->rgbtGreen;
ptr[2] = pix->rgbtBlue;
ptr[3] = 255;
}

break;
}
case 32:
{
// RGB 32 BITS
for( col = 0; col < w; col++, ptr += 4 )
{
// // convert bgr pixel (32 bits) into rgba (32 bits) pixel
RGBQUAD *pix = (RGBQUAD *)pBuff;
pBuff += sizeof( RGBQUAD );
ptr[0] = pix->rgbRed;
ptr[1] = pix->rgbGreen;
ptr[2] = pix->rgbBlue;
ptr[3] = 255;
}
break;
}
}
}
break;
}
```

```
case BI_RLE8:
{
// RLE 8 BITS
for( row = h - 1; row >= 0; row-- )
{
if( flipvert )
ptr = &(*pixels)[ row * w * 4 ];
for( col = 0; col < w; /* nothing */ )
{
// get one packet (2 bytes)
unsigned char byte1 = *((unsigned char *)(pBuff++));
unsigned char byte2 = *((unsigned char *)(pBuff++));
if( byte1 == RLE_COMMAND )
{
// absolute encoding
for( i = 0; i < byte2; i++, ptr += 4, col++ )
{
// read the current pixel
unsigned char color = *((unsigned char *)(pBuff++));
// convert indexed pixel (8 bits) into rgba (32 bits) pixel
ptr[0] = win_palette[ color ].rgbRed;
ptr[1] = win_palette[ color ].rgbGreen;
ptr[2] = win_palette[ color ].rgbBlue;
ptr[3] = 255;
}
if( (byte2 % 2) == 1 )
pBuff++;
}
else
{
// read next pixels
for( i = 0; i < byte1; i++, ptr += 4, col++ )
{
// convert indexed pixel (8 bits) into rgba (32 bits) pixel
ptr[0] = win_palette[ byte2 ].rgbRed;
ptr[1] = win_palette[ byte2 ].rgbGreen;
ptr[2] = win_palette[ byte2 ].rgbBlue;
ptr[3] = 255;
}
}
}
}
break;
}
case BI_RLE4:
{
// RLE 4 BITS
unsigned char color;
int   bytesRead = 0;           // number of bytes read
for( row = h - 1; row >= 0; row-- )
{
if( flipvert )
ptr = &(*pixels)[ row * w * 4 ];
for( col = 0; col < w; /* nothing */ )
{
// get one packet (2 bytes)
unsigned char byte1 = *((unsigned char *)(pBuff++));
unsigned char byte2 = *((unsigned char *)(pBuff++));
bytesRead += 2;
if( byte1 == RLE_COMMAND )
```

```
{
// absolute encoding
unsigned char databyte;
for( i = 0; i < byte2; i++, ptr += 4, col++ )
{
if( (i % 2) == 0 )
{
      // read the current pixel
      databyte = *((unsigned char *)(pBuff++));
      bytesRead++;

      color = (databyte >> 4);      // 4 first bits
}
else
{
      color = (databyte & 0x0F);    // 4 last bits
}
// convert indexed pixel (4 bits) into rgba (32 bits) pixel
ptr[0] = win_palette[ color ].rgbRed;
ptr[1] = win_palette[ color ].rgbGreen;
ptr[2] = win_palette[ color ].rgbBlue;
ptr[3] = 255;
}
while( (bytesRead % 2) != 0 )
{
pBuff++;
bytesRead++;
}
}
else
{
// read next pixels
for( i = 0; i < byte1; i++, ptr += 4, col++ )
{
if( (i % 2) == 0 )
      color = (byte2 >> 4);   // 4 first bits
else
      color = (byte2 & 0x0F); // 4 last bits
// convert indexed pixel (4 bits) into rgba (32 bits) pixel
ptr[0] = win_palette[ color ].rgbRed;
ptr[1] = win_palette[ color ].rgbGreen;
ptr[2] = win_palette[ color ].rgbBlue;
ptr[3] = 255;
}
}
}
}
break;
}
}
// free buffer memory
delete [] buffer;
// return success
return 1;
}

int LoadFilePCX( const char *filename, unsigned char **pixels, int *width, int
*height, bool flipvert )
{
```

```
FILE*        file;              // file stream
PCXHEADER               *header;          // header PCX
unsigned char           *data;                 // uncompressed paletted image
data
unsigned char           *ptr;             // pointer to pixels data
unsigned char           c;                     // temporary variable
char                    *buffer;          // buffer storing the entire file
int                           idx = 0;         // temporary variable
int                           numRepeat;       // temporary variable
int                           i, j;            // temporary variables

//////////////////////////////////////////////////////
// read the entire file in the buffer
file = fopen( filename, "rb" );
if( file == NULL )
return 0;
fseek(file, 0, SEEK_END);
long flen = ftell(file);
fseek(file, 0, SEEK_SET);
buffer = new char[ flen + 1 ];
fread(buffer, flen, 1, file);
char *pBuff = buffer;
fclose(file);
//////////////////////////////////////////////////////
header = (PCXHEADER *)pBuff;
if( (header->manufacturer     != 10)      ||
 (header->version        != 5) ||
 (header->encoding       != 1) ||
 (header->bitsPerPixel   != 8) )
{
return 0;
}
header->width      = header->width    - header->x + 1;
header->height     = header->height - header->y + 1;
if( width )
*width = header->width;
if( height )
*height = header->height;
if( !pixels )
{
delete [] buffer;
return (-1);
}
// allocate memory for image data
data = new unsigned char[ header->width * header->height ];
pBuff = (char *)&buffer[ 128 ];
// uncode compressed image (RLE)
while( idx < (header->width * header->height) )
{
if( (c = *(pBuff++)) > 0xbf )
{
numRepeat = 0x3f & c;
c = *(pBuff++);
for( i = 0; i < numRepeat; i++ )
data[ idx++ ] = c;
}
else
data[ idx++ ] = c;
}
```

```
// the palette is located at the 769th last byte of the file
pBuff = &buffer[ flen - 769 ];
// verify the palette; first char must be equal to 12
if( *(pBuff++) != 12 )
{
delete [] buffer;
delete [] data;
return 0;
}
// read the palette
header->palette = (unsigned char *)pBuff;
// allocate memory for 32 bits pixel data
*pixels = new unsigned char[ header->width * header->height * 4 ];
ptr = &(*pixels)[0];
// convert from paletted to 32 bits rgba pixels
for( j = header->height - 1; j > 0; j-- )
{
if( flipvert )
ptr = &(*pixels)[ j * header->width * 4 ];
for( i = 0; i < header->width; i++, ptr += 4 )
{
int color = 3 * data[ j * header->width + i ];
ptr[0] = (unsigned char)header->palette[ color + 0 ];
ptr[1] = (unsigned char)header->palette[ color + 1 ];
ptr[2] = (unsigned char)header->palette[ color + 2 ];
ptr[3] = (unsigned char)255;
}
}
delete [] data;
return 1;
}

int LoadFileTGA( const char *filename, unsigned char **pixels, int *width, int
*height, bool flipvert )
{
FILE* file;             // file stream
TGAHEADER         *tgah;                  // targa header
RGBTRIPLE         *palette;         // pointer on the color palette
char              *buffer;          // buffer storing the entire file
unsigned char     *ptr;             // pointer to pixels data
int                     row, col, i;      // temporary variables
/////////////////////////////////////////////////////
// read the entire file in the buffer
file = fopen( filename, "rb" );
if( file == NULL )
return 0;
fseek(file, 0, SEEK_END);
long flen = ftell(file);
fseek(file, 0, SEEK_SET);
buffer = new char[ flen + 1 ];
fread(buffer, flen, 1, file);
char *pBuff = buffer;
fclose(file);
// read the header
tgah = (TGAHEADER *)pBuff;
pBuff += sizeof( TGAHEADER );
if( width )
*width = tgah->is_width;
if( height )
*height = tgah->is_height;
```

```
if( !pixels )
{
delete [] buffer;
return (-1);
}
// allocate memory to store pixel data
*pixels     = new unsigned char[ tgah->is_width * tgah->is_height * 4 ];
ptr         = &(*pixels)[0];
// move the pixel data pointer to the begening of bitmap data
if( tgah->id_lenght )
pBuff = buffer + (tgah->id_lenght * sizeof( unsigned char ));
/////////////////////////////////////////////////////
// read the palette
if( tgah->color_map_type )
{
// 24 and 32 bits images are not paletted
// ajust the palette pointer to the memory in the buffer
palette = (RGBTRIPLE *)pBuff;
pBuff += tgah->cm_length * (tgah->cm_size >> 3) * sizeof( unsigned char );
}
/////////////////////////////////////////////////////
// read pixel data following the image compression
// type and the number of bits per pixels
/////////////////////////////////////////////////////
switch( tgah->image_type )
{
case 0:
// pas de données image
break;
case 1:
// COLOR-MAPPED BGR 8 BITS GREYSCALE
case 3:
{
// COLOR-MAPPED BGR 8 BITS
for( row = tgah->is_height - 1; row >= 0; row-- )
{
if( flipvert )
ptr = &(*pixels)[ row * tgah->is_width * 4 ];
for( col = 0; col < tgah->is_width; col++, ptr += 4 )
{
// read the current pixel
unsigned char color = *((unsigned char *)(pBuff++));
// convert indexed pixel (8 bits) into rgba (32 bits) pixel
ptr[0] = palette[ color ].rgbtRed;        // b->r
ptr[1] = palette[ color ].rgbtGreen;      // g->g
ptr[2] = palette[ color ].rgbtBlue;       // r->b
ptr[3] = 255;                                         // alpha
}
}
break;
}
case 2:
{
for( row = tgah->is_height - 1; row >= 0; row-- )
{
if( flipvert )
ptr = &(*pixels)[ row * tgah->is_width * 4 ];
for( col = 0; col < tgah->is_width; col++, ptr += 4 )
{
switch( tgah->is_pixel_depth )
{
```

```
case 16:
{
// TRUE-COLOR BGR 16 BITS
// read the current pixel
unsigned short color = *((unsigned short *)pBuff);
pBuff += sizeof( short );
// convert bgr (16 bits) pixel into rgba (32 bits) pixel
ptr[0] = ((color & 0x7C00) >> 10) << 3;   // b->r
ptr[1] = ((color & 0x03E0) >>  5) << 3;   // g->g
ptr[2] = ((color & 0x001F) >>  0) << 3;   // r->b
ptr[3] = 255;                                         // alpha
break;
}
case 24:
{
// TRUE-COLOR BGR 24 BITS
// convert bgr (24 bits) pixel into rgba (32 bits) pixel
RGBTRIPLE *pix = (RGBTRIPLE *)pBuff;
pBuff += sizeof( RGBTRIPLE );
ptr[0] = pix->rgbtRed;
ptr[1] = pix->rgbtGreen;
ptr[2] = pix->rgbtBlue;
ptr[3] = 255;
break;
}
case 32:
{
// TRUE-COLOR BGR 32 BITS
// convert bgr (32 bits) pixel into rgba (32 bits) pixel
BGRAQUAD *pix = (BGRAQUAD *)pBuff;
pBuff += sizeof( BGRAQUAD );
ptr[0] = pix->bgraRed;
ptr[1] = pix->bgraGreen;
ptr[2] = pix->bgraBlue;
ptr[3] = pix->bgraAlpha;
break;
}
}
}
}
break;
}
case 9:
// RLE COLOR-MAPPED BGR 8 BITS
case 11:
{
// RLE COLOR-MAPPED BGR 8 BITS GREYSCALE
unsigned char     packetHeader, packetSize, i;
for( row = tgah->is_height - 1; row >= 0; row-- )
{
if( flipvert )
ptr = &(*pixels)[ row * tgah->is_width * 4 ];
for( col = 0; col < tgah->is_width; /* rien */ )
{
packetHeader      = *((unsigned char *)(pBuff++));
packetSize        = 1 + (packetHeader & 0x7f);
if( packetHeader & 0x80 )
{
// run-length packet
// read the current pixel
unsigned char color = *((unsigned char *)(pBuff++));
```

```
// convert indexed pixel (8 bits) pixel into rgba (32 bits) pixel
for( i = 0; i < packetSize; i++, ptr += 4, col++ )
{
ptr[0] = palette[ color ].rgbtRed;       // b->r
ptr[1] = palette[ color ].rgbtGreen;     // g->g
ptr[2] = palette[ color ].rgbtBlue;      // r->b
ptr[3] = 255;                                        // alpha
}
}
else
{
// non run-length packet
for( i = 0; i < packetSize; i++, ptr += 4, col++ )
{
// read the current pixel
unsigned char color = *((unsigned char *)(pBuff++));
// convert indexed pixel (8 bits) pixel into rgba (32 bits) pixel
ptr[0] = palette[ color ].rgbtRed;       // b->r
ptr[1] = palette[ color ].rgbtGreen;     // g->g
ptr[2] = palette[ color ].rgbtBlue;      // r->b
ptr[3] = 255;                                        // alpha
}
}
}
}
break;
}
case 10:
{
unsigned char     packetHeader, packetSize;
for( row = tgah->is_height - 1; row >= 0; row-- )
{
if( flipvert )
ptr = &(*pixels)[ row * tgah->is_width * 4 ];
for( col = 0; col < tgah->is_width; /* rien */ )
{
packetHeader      = *((unsigned char *)(pBuff++));
packetSize        = 1 + (packetHeader & 0x7f);
if( packetHeader & 0x80 )
{
// run-length packet
switch( tgah->is_pixel_depth )
{
case 16:
{
// RLE TRUE-COLOR BGR 16 BITS
      // read the current pixel
      unsigned short color = *((unsigned short *)pBuff);
      pBuff += sizeof( short );
      // convert bgr (16 bits) pixel into rgba (32 bits) pixel
      for( i = 0; i < packetSize; i++, ptr += 4, col++ )
      {
            ptr[0] = ((color & 0x7C00) >> 10) << 3;   // b->r
            ptr[1] = ((color & 0x03E0) >>  5) << 3;   // g->g
            ptr[2] = ((color & 0x001F) >>  0) << 3;   // r->b
            ptr[3] = 255;
      }
      break;
}
```

```
case 24:
{
// RLE TRUE-COLOR BGR 24 BITS
      // convert bgr (24 bits) pixel into rgba (32 bits) pixel
      RGBTRIPLE *pix = (RGBTRIPLE *)pBuff;
      pBuff += sizeof( RGBTRIPLE );
      for( i = 0; i < packetSize; i++, ptr += 4, col++ )
      {
            ptr[0] = pix->rgbtRed;
            ptr[1] = pix->rgbtGreen;
            ptr[2] = pix->rgbtBlue;
            ptr[3] = 255;
      }
      break;
}
case 32:
{
// RLE TRUE-COLOR BGR 32 BITS
      // convert bgr (32 bits) pixel into rgba (32 bits) pixel
      BGRAQUAD *pix = (BGRAQUAD *)pBuff;
      pBuff += sizeof( BGRAQUAD );
      for( i = 0; i < packetSize; i++, ptr += 4, col++ )
      {
            ptr[0] = pix->bgraRed;
            ptr[1] = pix->bgraGreen;
            ptr[2] = pix->bgraBlue;
            ptr[3] = pix->bgraAlpha;
      }
      break;
}
}
}
else
{
// non run-length packet
for( i = 0; i < packetSize; i++, ptr += 4, col++ )
{
switch( tgah->is_pixel_depth )
{
      case 16:
      {
// RLE TRUE-COLOR BGR 16 BITS
            // read the current pixel
            unsigned short color = *((unsigned short *)pBuff);
            pBuff += sizeof( short );
            // convert bgr (16 bits) pixel into rgba (32 bits) pixel
            ptr[0] = ((color & 0x7C00) >> 10) << 3;   // b->r
            ptr[1] = ((color & 0x03E0) >>  5) << 3;   // g->g
            ptr[2] = ((color & 0x001F) >>  0) << 3;   // r->b
            ptr[3] = 255;                                     // alpha
            break;
      }
      case 24:
      {
// RLE TRUE-COLOR BGR 24 BITS
            // convert bgr (24 bits) pixel into rgba (32 bits) pixel
            RGBTRIPLE *pix = (RGBTRIPLE *)pBuff;
            pBuff += sizeof( RGBTRIPLE );
            ptr[0] = pix->rgbtRed;
            ptr[1] = pix->rgbtGreen;
            ptr[2] = pix->rgbtBlue;
```

```
case 24:
{
// RLE TRUE-COLOR BGR 24 BITS
      // convert bgr (24 bits) pixel into rgba (32 bits) pixel
      RGBTRIPLE *pix = (RGBTRIPLE *)pBuff;
      pBuff += sizeof( RGBTRIPLE );
      for( i = 0; i < packetSize; i++, ptr += 4, col++ )
      {
            ptr[0] = pix->rgbtRed;
            ptr[1] = pix->rgbtGreen;
            ptr[2] = pix->rgbtBlue;
            ptr[3] = 255;
      }
      break;
}
case 32:
{
// RLE TRUE-COLOR BGR 32 BITS
      // convert bgr (32 bits) pixel into rgba (32 bits) pixel
      BGRAQUAD *pix = (BGRAQUAD *)pBuff;
      pBuff += sizeof( BGRAQUAD );
      for( i = 0; i < packetSize; i++, ptr += 4, col++ )
      {
            ptr[0] = pix->bgraRed;
            ptr[1] = pix->bgraGreen;
            ptr[2] = pix->bgraBlue;
            ptr[3] = pix->bgraAlpha;
      }
      break;
}
}
}
else
{
// non run-length packet
for( i = 0; i < packetSize; i++, ptr += 4, col++ )
{
switch( tgah->is_pixel_depth )
{
      case 16:
      {
// RLE TRUE-COLOR BGR 16 BITS
            // read the current pixel
            unsigned short color = *((unsigned short *)pBuff);
            pBuff += sizeof( short );
            // convert bgr (16 bits) pixel into rgba (32 bits) pixel
            ptr[0] = ((color & 0x7C00) >> 10) << 3;   // b->r
            ptr[1] = ((color & 0x03E0) >>  5) << 3;   // g->g
            ptr[2] = ((color & 0x001F) >>  0) << 3;   // r->b
            ptr[3] = 255;                                   // alpha
            break;
      }
      case 24:
      {
// RLE TRUE-COLOR BGR 24 BITS
            // convert bgr (24 bits) pixel into rgba (32 bits) pixel
            RGBTRIPLE *pix = (RGBTRIPLE *)pBuff;
            pBuff += sizeof( RGBTRIPLE );
            ptr[0] = pix->rgbtRed;
            ptr[1] = pix->rgbtGreen;
            ptr[2] = pix->rgbtBlue;
```

```
            ptr[3] = 255;
            break;
      }
      case 32:
      {
// RLE TRUE-COLOR BGR 32 BITS
            // convert bgr (32 bits) pixel into rgba (32 bits) pixel
            BGRAQUAD *pix = (BGRAQUAD *)pBuff;
            pBuff += sizeof( BGRAQUAD );
            ptr[0] = pix->bgraRed;
            ptr[1] = pix->bgraGreen;
            ptr[2] = pix->bgraBlue;
            ptr[3] = pix->bgraAlpha;

            break;
      }
}
}
}
}
}
break;
}
default:
{
// unknown format
delete [] pixels;
delete [] buffer;
return 0;
}
}
delete [] buffer;
return 1;
}


//////////////////////////////////////////////////////////////////////
unsigned int LoadTexture(char *filename)
{
unsigned int      id = 0;
unsigned char     *texels = 0;
int                     width, height;
int                     success;

if( strstr( filename, ".bmp" ) || strstr( filename, ".BMP" ) )
success = LoadFileBMP( filename, &texels, &width, &height, true );
if( strstr( filename, ".tga" ) || strstr( filename, ".TGA" ) )
success = LoadFileTGA( filename, &texels, &width, &height, true );
if( strstr( filename, ".pcx" ) || strstr( filename, ".PCX" ) )
success = LoadFilePCX( filename, &texels, &width, &height, true );
if( success > 0 )
{
// create and initialize new texture
glGenTextures( 1, &id );
glBindTexture( GL_TEXTURE_2D, id );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT );
gluBuild2DMipmaps( GL_TEXTURE_2D, GL_RGBA, width, height, GL_RGBA,
GL_UNSIGNED_BYTE, texels );
```

```
}
else
{
id = NULL;
}
if( texels )
delete [] texels;
return id;
}
```

إذا كنت تعتقد أن الكود السابق طويل و صعب فأنت مخطئ, هو كود طويل و سهل و كل ما يفعله هو قراءة بايتات الصور و تحميلها إلى إكساء و لكن ليس موضوعنا اليوم تركيبة ملفات bmp و tga و pcx لذلك لا تعليق على الكود فكثيرة هي المصادر التي تشرح بنية ملفات الصور.

نعود الآن إلى موضوعنا الأساسي و هو موديلات MD2, ككل أنواع الملفات, تتكون ملفات md2 من قسمين, رأس الملف و بياناته, رأس الملف به معلومات حول إصدار الموديل و حجم و عدد المشاهد و طول و ارتفاع الإكساء, عناوين إحداثيات الجسم و إحداثيات إكساءه...الخ.

سنقوم بتطوير كلاس نسميها CMD2Model و تكون مسؤولة عن تحميل و عرض و تحريك الموديل و في نفس الوقت أشرح بنية ملفات MD2 , لذلك أضف كلاس جديدة لمشروع بإسم CMD2Model و بذلك نحصل على ملفين اثنين: MD2Model.h و MD2Model.cpp, الكود الموجود في الملف MD2Model.h هو :

```
#pragma once
#ifndef         __MD2MODEL_H_
#define         __MD2MODEL_H_
class CMD2Model
{
public:
      CMD2Model(void);
      ~CMD2Model(void);
};
#endif //__MD2MODEL_H_
```

أما الملف MD2Model.cpp فيحوي هذا الكود :

```
#include "MD2Model.h"

CMD2Model::CMD2Model(void)
{
}

CMD2Model::~CMD2Model(void)
{
}
```

بسم الله, من هنا نبدأ, قلنا أن ملف MD2 يحتوي على رأس و بيانات فإنه يجب علينا تعريف بنية الرأس و ذلك على مستوى الملف MD2Model.h:

```
#pragma once
#ifndef            __MD2MODEL_H_
#define            __MD2MODEL_H_
/* vector */
typedef float vec3_t[3]; //
/* md2 header */
typedef struct
{
    int ident; //IP2D معرف الملف, يجب أن يكون
    int version; //إصدار الملف

    int skinwidth; //عرض الإكساء المستخدم
    int skinheight; //إرتفاع الإكساء

    int framesize; //حجم المشهد

    int num_skins; //عدد الإكساءات
    int num_vertices; //عدد النقاط المكونة للموديل
    int num_st; //عدد النقاط المستخدمة للإكساء
    int num_tris; //عدد المثلثات ال مكونة للموديل
    int num_glcmds; //أوامر أوبنجل المستخدمة لرسم الموديل
    int num_frames; //عدد المشاهد و التي ستكون الحركة

    int offset_skins; //عنوان أسماء الإكساءات في الملف
    int offset_st; //عنوان إحداثيات الإكساء
    int offset_tris; //عنوان إحداثيات المثلثات
    int offset_frames; //عنوان أول مشهد
    int offset_glcmds; //عموان أوامر الأوبنجل
    int offset_end; //عنوان إلى نهاية الملف و لن نحتاجه

} md2_header_t;
//...
#endif //__MD2MODEL_H_
```

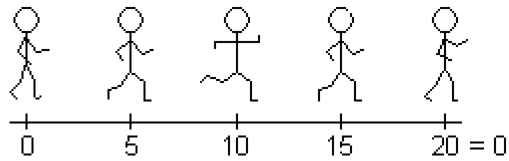
لنلقي نظرة على الأشياء الهامة:

framessize سيحمل حجم كل مشهد بالبايت, المشهد هو الصورة المكونة للحركة أو الفلم, بتعرض عدة مشاهد بسرعة معينة نحصل على حركة, لذلك ملف MD2 يتكون من 199 مشهد مقسمة على 21 حركة, يتكون المشهد من قائمة لإحداثيات النقاط المكونة للمثلثات التي تكون سطح الموديل أو الجسم ثلاثي الأبعاد.

num_glcmds يحمل عدد أوامر OpenGL المستخدمة لرسم الموديل, قائمة أوامر OpenGL هي قائمة لأعداد من نوع int والتي تحدد نوع المثلثات الذي سنستخدمه لرسم الموديل, فإذا كان الرقم الحالي سالب فإننا نستخدم GL_TRIANGLE_FAN وإذا كان موجب نستخدم GL_TRIANGLE_STRIP و هذا طبعا كبارامتر للدالة glBegin().

أول ما سنحتاجه هو تعريف نوع جديد من البيانات و هو الشعاع, و بما أن الشعاع يتحدد بثلاث إحداثيات x,y,z فإننا سنعرفه على شكل مصفوفة من ثلاث عناصر و قد عرفناه قبل أن نعرف تركيبة رأس ملف MD2 و أعطيناه اسم vec3_t.

num_frames يحمل العدد الكلي للمشاهد المكونة لحركة الموديل, ولكن, حتي حركة بسيطة تتكون من 200 أو 300 مشهد فمن المستحيل حفظ كل المشاهد, لذلك سنحفظ فقط المشاهد المفتاحية و نقوم بحساب المشاهد التي تتوسط المشاهد المفتاحية أثناء رسم الموديل و هذا باستعمال linear interpolation (أشرحه لاحقا), لاحظ هذه الصورة:



المشاهد المفتاحية

تمثل تلك الصورة موديل بحركة الجري المتكونة من 20 مشهد و لكن سنحتفظ فقط بخمس مشاهد و هي 0 5 10 15 20, أما المشاهد الأخرى فيجب حسابها حتى تتحصل على حركة سلسة.

يتكون سطح الموديل 3D من نقاط القمم و موصولة بمثلثات و لكل قمة ثلاث إحداثيات x,y,z و شعاع normal و هو شعاع يحدد إلى أي جهة تتجه القمة من أجل إضاءة صحيحة, لذلك نعرف هذه التركيبة التي تمثل القمة:

```
#pragma once
#ifndef         __MD2MODEL_H_
#define         __MD2MODEL_H_

typedef struct
{
  unsigned char v[3];
  unsigned char normalIndex;

} md2_vertex_t;
//...
#endif //__MD2MODEL_H_
```

المصفوفة v[3] تحمل ثلاث قيم تحدد ثلاث إحداثيات على ثلاث محاور, ولكن تلاحظ أنها من نوع BYTEأي أنها تتراوح من 0 إلى 255 و هذا لأنها مضغوطة و لفك ضغطها سنستعمل بعض البيانات الأخرى الخاصة بكل مشهد.

العنصر normalIndex من التركيبة يمثل مؤشر إلى شعاع محدد في مصفوفة معرفة مسبقا و تحمل قيم ال normals الخاصة بموديلات MD2 و طبعا هذا للحصول على إضاءة صحيحة.

أيضا سنحتاج إل تركيبة أخرى و هي :

```
#pragma once
#ifndef         __MD2MODEL_H_
#define         __MD2MODEL_H_

typedef struct
{
  short s;
  short t;

} md2_texCoord_t;
//...
#endif //__MD2MODEL_H_
```

تحدد هذه التركيبة إحداثيات الإكساء المرافقة لكل قمة و هي مضغوطة طبعا إذ أنها من النوع short عوض float, ولحساب الإحداثيات الحقيقية للإكساء نستعمل هذه المعادلة:

RealST[i].s = (float)texCoord[i].s / header.skinwidth;
RealST[i].t = (float)texCoord[i].t / header.skinheigh

وكل مشهد سيتكون من هذه التركيبة :

```
#pragma once
#ifndef          __MD2MODEL_H_
#define          __MD2MODEL_H_

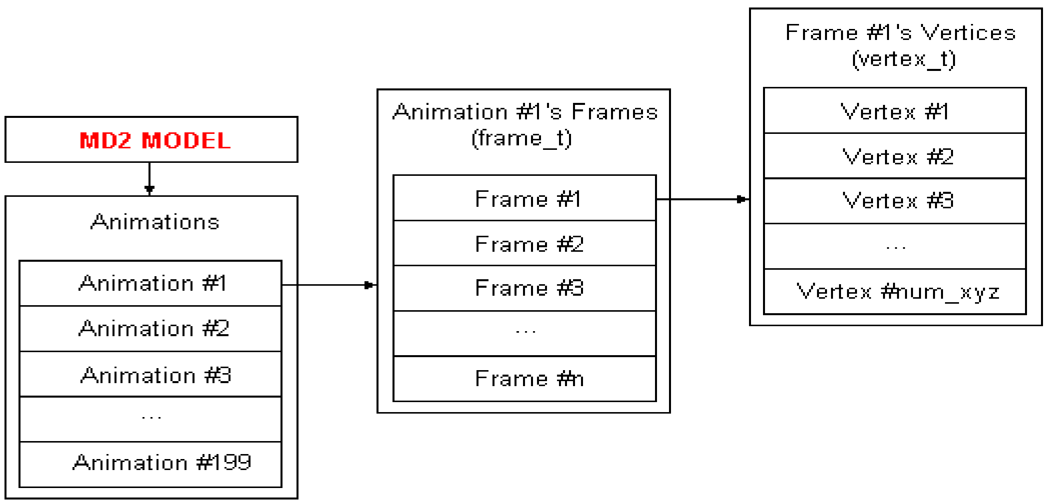
typedef struct
{
    vec3_t scale;
    vec3_t translate;
    char name[16]; //إسم المشهد
    md2_vertex_t *verts; //قائمة قمم المشهد
} md2_frame_t;
//...
#endif //__MD2MODEL_H_
```

قلنا سابقا أن الإحداثيات الموجودة في التركيبة md2_vertex_t مضغوطة و لفك ضغطها نستعمل بيانات خاصة بكل مشهد, هذه البيانات هي scale و translate الموجودة في تركيبة كل مشهد, ولفك ضغط الإحداثية فإننا نضربها في scaleو نضيف إليها translate :

*vertex.x = (frame.verts[i].v[0] * frame.scale[0]) + frame.translate[0]*
*vertex.y = (frame.verts[i].v[1] * frame.scale[1]) + frame.translate[1]*
*vertex.z = (frame.verts[i].v[2] * frame.scale[2]) + frame.translate[2]*

أما العنوان verts فهو عنوان أول قمة في المشهد, أي أن:

```
frame.verts[ 2 ] // ثاني قمة في المشهد
frame.verts[ i ] // القمة إي في المشهد
frame.verts[ num_xyz - 1 ] // آخر قمة في المشهد
```



- بنية ملف MD2 -

ولربط كل قمة من قمم المجسم أو الموديل بإحداثيات الإكساء الموافقة لها نضيف هذه البنية :

```
#pragma once
#ifndef         __MD2MODEL_H_
#define         __MD2MODEL_H_

typedef struct
{
  unsigned short vertex[3]; //مؤشر إلى قمة من قمم المثلث
  unsigned short st[3]; //مؤشر إلى إحداثيات الإكساء الموافقة للقمة
} md2_triangle_t;
//...
#endif //__MD2MODEL_H_
```

ولحفظ اسم الإكساء ننشئ هذه التركيبة فقط للتنظيم فنحن لا نحتاج إلى مسار الإكساء:

```
#pragma once
#ifndef         __MD2MODEL_H_
#define         __MD2MODEL_H_

typedef struct
{
  char name[64];
} md2_skin_t;
//...
#endif //__MD2MODEL_H_
```

نعرف الآن البنية التي تشمل كل ما نحتاجه لحفظ بيانات الموديل و هي:

```
#pragma once
#ifndef         __MD2MODEL_H_
#define         __MD2MODEL_H_

typedef struct
{
  md2_header_t header;
  md2_skin_t *skins;
  md2_texCoord_t *texcoords;
  md2_triangle_t *triangles;
  md2_frame_t *frames;
  int *glcmds;
  unsigned int tex_id;
} md2_model_t;
//...
#endif //__MD2MODEL_H_
```

ليس فيها أي جديد, هي عبارة عن تركيبة لجمع التركيبات السابقة فقط.

قبل أن نبدأ في قراءة و عرض حركة الموديل نكتب الكود الخاص بباني الكلاس و هادم الكلاس, نضيف إلى الكلاس كل ما نحتاجه من دوال و متغيرات حتى ننتقل نهائيا إلى الملف CMD2Model.cpp كتابة أجسام الدوال, على مستوى الملف CMD2Model.h نعدل الكلاس لتصبح:

```
#pragma once
#ifndef             __MD2MODEL_H_
#define             __MD2MODEL_H_
//...
class CMD2Model
{
public:
      md2_model_t *mdl; //متغير من نوع التركيبة التي تمثل الموديل
public:
      CMD2Model(void);
      ~CMD2Model(void);
      int ReadMD2Model(char *filename); //تحميل الموديل
      void ReadTexture(char *filename); //تحميل الإكساء
      void RenderFrame(int n, float interp); //رسم مشهد
      void Animate(int start, int end, int *frame, float *interp); // الدالة التي نحدد بها سرعة رسم المشاهد (مشهد في الثانية) و ننتقل بها من مشهد لإخر
      void FreeModel(); //تحرير الذاكرة المستغلة من طرف الكلاس
      void AllocateModel(); //حجز ذاكرة جديدة لتحميل موديل جديد
};
#endif //__MD2MODEL_H_
```

ننتقل الآن إلى الملف CMD2Model.cpp لتعريف أجسام الدوال و نبدأ بالباني و الهادم + الدالة التي تحجز الذاكرة مع التي تحرر الذاكرة و هذه أساسيات السي بلس بلس:

```
#include "MD2Model.h"
#include <GL/glut.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "textures.h"

CMD2Model::CMD2Model(void)
{
       mdl = (md2_model_t *)malloc (sizeof (md2_model_t));
}

CMD2Model::~CMD2Model(void)
{
   if (mdl)
     {
       FreeModel();
     }
}

void CMD2Model::AllocateModel()
{
      mdl = (md2_model_t *)malloc (sizeof (md2_model_t));
}

void CMD2Model::FreeModel()
{
  int i;
```

```
  if (mdl->skins)
    {
      free (mdl->skins);
      mdl->skins = NULL;
    }

  if (mdl->texcoords)
    {
      free (mdl->texcoords);
      mdl->texcoords = NULL;
    }

  if (mdl->triangles)
    {
      free (mdl->triangles);
      mdl->triangles = NULL;
    }

  if (mdl->glcmds)
    {
      free (mdl->glcmds);
      mdl->glcmds = NULL;
    }

  if (mdl->frames)
    {
      for (i = 0; i < mdl->header.num_frames; ++i)
            {
            free (mdl->frames[i].verts);
            mdl->frames[i].verts = NULL;
            }

      free (mdl->frames);
      mdl->frames = NULL;
    }
  free (mdl);
  mdl = NULL;
}
```

كيف نحمل الموديل؟ ... هذا ما تقوم به الدالة ReadMD2Model:

```
//...
int CMD2Model::ReadMD2Model(char *filename)
{
  FILE *fp;
  int i;

  fp = fopen (filename, "rb");
  if (!fp)  //نتأكد من أننا فتحنا الملف
    {
      printf ("error: couldn't open \"%s\"!", filename);
      return 0;
    }
  /* نقرأ رأس الملف */
  fread (&mdl->header, 1, sizeof (md2_header_t), fp);
  if ((mdl->header.ident != 844121161) ||       //نتحقق من معرف الملف
      (mdl->header.version != 8))          //نتحقق من إصدار الموديل
    {
```

```
        printf ("error: bad version!");
        fclose (fp);
        return 0;
    }
  /* نحجز الذاكرة المناسبة لكل بنية */
  mdl->skins = (md2_skin_t *)malloc (sizeof (md2_skin_t) *
        mdl->header.num_skins);
  mdl->texcoords = (md2_texCoord_t *)malloc (sizeof (md2_texCoord_t) *
        mdl->header.num_st);
  mdl->triangles = (md2_triangle_t *)malloc (sizeof (md2_triangle_t) *
        mdl->header.num_tris);
  mdl->frames = (md2_frame_t *)malloc (sizeof(md2_frame_t) *
        mdl->header.num_frames);
  mdl->glcmds = (int *)malloc (sizeof (int) * mdl->header.num_glcmds);
  /* نحمل المعلومات إلى البنية المناسبة */
  fseek (fp, mdl->header.offset_skins, SEEK_SET);
  fread (mdl->skins, sizeof (md2_skin_t), mdl->header.num_skins, fp);

  fseek (fp, mdl->header.offset_st, SEEK_SET);
  fread (mdl->texcoords, sizeof (md2_texCoord_t), mdl->header.num_st, fp);

  fseek (fp, mdl->header.offset_tris, SEEK_SET);
  fread (mdl->triangles, sizeof (md2_triangle_t), mdl->header.num_tris, fp);

  fseek (fp, mdl->header.offset_glcmds, SEEK_SET);
  fread (mdl->glcmds, sizeof (int), mdl->header.num_glcmds, fp);
  /* نقرأ المشاهد */
  fseek (fp, mdl->header.offset_frames, SEEK_SET);
  for (i = 0; i < mdl->header.num_frames; ++i)
    {
      /* حجز الذاكرة اللازمة لحفظ قمم المشهد */
      mdl->frames[i].verts = (md2_vertex_t *)
      malloc (sizeof (md2_vertex_t) * mdl->header.num_vertices);
      /* قراءة البيانات */
      fread (mdl->frames[i].scale, sizeof (vec3_t), 1, fp);
      fread (mdl->frames[i].translate, sizeof (vec3_t), 1, fp);
      fread (mdl->frames[i].name, sizeof (char), 16, fp);
      fread (mdl->frames[i].verts, sizeof (md2_vertex_t),
             mdl->header.num_vertices, fp);
    }

  fclose (fp);
  return 1;
}
```

أصبح الموديل جاهزا, ماذا بعد؟... نحمل الإكساء الخاص بكل موديل عن طريق الدالة :

```
//...
void CMD2Model::ReadTexture(char *filename)
{
     mdl->tex_id = LoadTexture(filename);
}
```

دالة بسيطة و مفهومة...

حان وقت رسم الموديل, سنرسمه باستخدام قائمة OpenGL commands المحفوظة في ملف الموديل نفسه, كيف تعمل و ماذا تحدد.

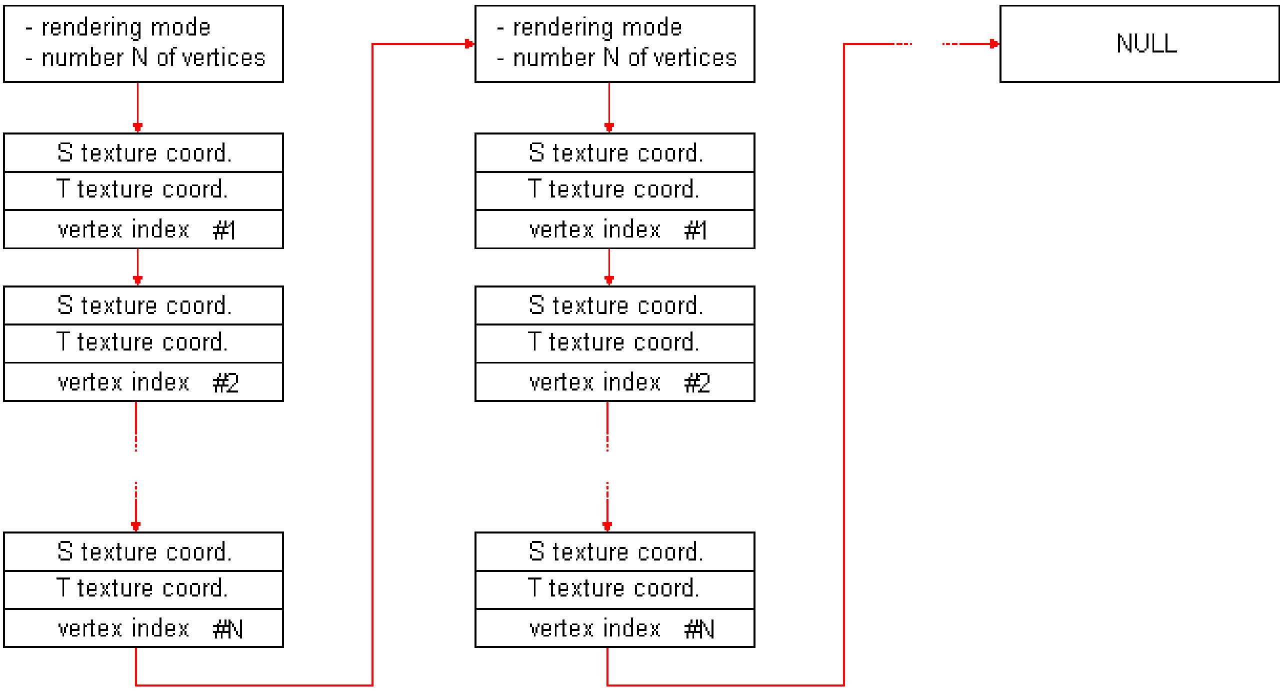
سنضع مؤشرا يشير إلى بداية القائمة – أي أنه سيشير إلى offset_glcmds – ونقرأ كل الأومر إلى أن يعيد لنا المؤشر القيمة 0, و هي آخر قيمة في قائمة أوامر OpenGL, ماذا بعد؟

نقرأ القيمة الأولى و تحدد شيئين: نوع المثلثات الذي سنستخدمه لرسم النقاط المواليةGL_TRIANGLE_STRIP) إذا كانت القيمة موجبة أو GL_TRIANGLE_FAN إذا كانت القيمة سالبة), و تحدد عدد النقاط التي سنرسمها باستعمال ذلك النوع من المثلثات.

القيمتين التاليتين (s,t) إحداثيات الإكساء .

القيمة الأخيرة هي مؤشر للنقطة أو القمة نفسها التي سنرسمها.

بعد معالجة قمم كل المجموعة ننتقل إلى المجموعة التالية و هكذا إلى أن يعيد مؤشر قائمة أوامر OpenGL القيمة الخالية NULL.



نعود إلى الملف CMD2Model.h ونعرف التركيبة التي تحمل الثلاث متغيرات المقروءة عند كل أمر OpenGL:

```
//...
typedef struct
{
  float s;
  float t;
  int index;

} md2_glcmd_t;
//...
#endif //__MD2MODEL_H_
```

عودة إلى CMD2Model.cpp لنعرف الدالة RenderFrame التي ترسم مشهد ما:

```
//...
void CMD2Model::RenderFrame(int n, float interp)
{
  vec3_t anorms_table[162] = {
      #include "anorms.h"
      };
  int i, *pglcmds;
  vec3_t v_curr, v_next, v, norm;
  float *n_curr, *n_next;
  md2_frame_t *pframe1, *pframe2;
  md2_vertex_t *pvert1, *pvert2;
  md2_glcmd_t *packet;

  if ((n < 0) || (n > mdl->header.num_frames - 1))
    return;

  glBindTexture (GL_TEXTURE_2D, mdl->tex_id);
  /*تحدد بداية قائمة أوامر الأوبنجل pglcmds*/
  pglcmds = mdl->glcmds;
  /* نرسم الموديل */
  while ((i = *(pglcmds++)) != 0) // المتغير i يمثل عدد النقاط التي سنرسمها
    {
      if (i < 0)
      {
        glBegin (GL_TRIANGLE_FAN);
        i = -i;
      }
      else
      {
        glBegin (GL_TRIANGLE_STRIP);
      }
      /* نرسم كل قمة من المشهد */
      for (/* nothing */; i > 0; --i, pglcmds += 3)
      {
        packet = (md2_glcmd_t *)pglcmds;
        pframe1 = &mdl->frames[n];
        pframe2 = &mdl->frames[n + 1];
        pvert1 = &pframe1->verts[packet->index];
        pvert2 = &pframe2->verts[packet->index];
        /* نحدد إحداثيات الإكساء*/
        glTexCoord2f (packet->s, packet->t);
        /* نستخرج الـ normals*/
        n_curr = anorms_table[pvert1->normalIndex];
        n_next = anorms_table[pvert2->normalIndex];

        norm[0] = n_curr[0] + interp * (n_next[0] - n_curr[0]);
        norm[1] = n_curr[1] + interp * (n_next[1] - n_curr[1]);
        norm[2] = n_curr[2] + interp * (n_next[2] - n_curr[2]);

        glNormal3fv (norm);

        /* interpolate vertices */
        v_curr[0] = pframe1->scale[0] * pvert1->v[0] + pframe1->translate[0];
        v_curr[1] = pframe1->scale[1] * pvert1->v[1] + pframe1->translate[1];
        v_curr[2] = pframe1->scale[2] * pvert1->v[2] + pframe1->translate[2];

        v_next[0] = pframe2->scale[0] * pvert2->v[0] + pframe2->translate[0];
        v_next[1] = pframe2->scale[1] * pvert2->v[1] + pframe2->translate[1];
```

```
        v_next[2] = pframe2->scale[2] * pvert2->v[2] + pframe2->translate[2];

        v[0] = v_curr[0] + interp * (v_next[0] - v_curr[0]);
        v[1] = v_curr[1] + interp * (v_next[1] - v_curr[1]);
        v[2] = v_curr[2] + interp * (v_next[2] - v_curr[2]);

        glVertex3fv (v);
      }

      glEnd ();
    }
}
```

تلك الدالة تأخذ بارامترين: الأول هو n وهو يحدد المشهد الذي سنرسمه, قد يكون المشهد الأول, العاشر...الخ, أما البارمتر الثاني فنستخدمه و كأنه الوقت الذي نرسم فيه المشهد, مثلا إذا كان المشهد الأول هو إنسان واقف و المشهد الثاني هو إنسان جالس, عندما نرسم المشهد الأول في اللحظة 0 سنرسم الإنسان واقف, و في اللحظة 1 سنرسم الإنسان يستعد للجلوس و هكذا.... فقد قنا في بداية الدرس أن الحركة الواحدة يلزمها مئات المشاهد و قلنا أننا نخزن في ملف الـ MD2 المشاهد المفتاحية فقط أما المشاهد الأخرى فنحسبها بناءا على المشهد الحالي و المشهد التالي... هنا و باستخدام البارامتر interp نحسب و نرسم المشاهد الفرعية التي يجب أن تكون بين المشاهد المفتاحية.

قمنا بالتصريح عن متغيرين من نوع md2_frame_t وهما pframe1 و pframe2 و يمثلان على الترتيب المسهد الحالي و المشهد التالي, تقريبا لن نرسم النقطة لا التي تنتمي إلى المشهد الحالي و لا التي تنتمي إلى المشهد التالي, و إنما النقطة التي بينهما استنادا إلى الزمن أو قيمة المتغير interp, لذلك صرحنا على متغيرين pvert1 و pvert2 من نوع md2_vertex_t, نفس الشيء بالنسبة لـ Normals, نصرح عن ثلاث متغيرات من نوع vec3_t , الأول هو v_curr والذي يمثل النورمال للنقطة الحالية من المشهد الحالي, الثاني هو v_next والذي يمثل النورمال الذي يوافق النقطة الحالية من المشهد التالي, أما الثالث فهو norm والذي سيحمل القيمة المحسوبة بينهما كما يلي:

```
norm[0] = n_curr[0] + interp * (n_next[0] - n_curr[0]);
norm[1] = n_curr[1] + interp * (n_next[1] - n_curr[1]);
norm[2] = n_curr[2] + interp * (n_next[2] - n_curr[2]);
```

إذا وكما تلاحظ فإن سرعة تقدم حساب النورمال ومتعلقة بالمتغير interp والذي يحاكي الزمن.

قيم النورمال الموافقة لنقاط الموديل غير محفوظة داخل ملف MD2, كنا قد قلنا سابقا أنها محسوبة مسبقا, أي أننا سنأخذها على شكل مصفوفة ذات قيم معروفة, في الكود السابق صرحنا عن مصفوفة باسم anorms_table ذات 162 عنصرا ومن نوع vec3_t, وأعطيناها قيم المصفوفة الموجودة في الملف anorms.h, لذلك, أصف ملف جديد باسم anorms.h و اكتب فيه قيم النورمال كما يلي:

```
{ -0.525731f,  0.000000f,  0.850651f }, { -0.442863f,  0.238856f,  0.864188f },
{ -0.295242f,  0.000000f,  0.955423f }, { -0.309017f,  0.500000f,  0.809017f },
{ -0.162460f,  0.262866f,  0.951056f }, {  0.000000f,  0.000000f,  1.000000f },
{  0.000000f,  0.850651f,  0.525731f }, { -0.147621f,  0.716567f,  0.681718f },
{  0.147621f,  0.716567f,  0.681718f }, {  0.000000f,  0.525731f,  0.850651f },
{  0.309017f,  0.500000f,  0.809017f }, {  0.525731f,  0.000000f,  0.850651f },
{  0.295242f,  0.000000f,  0.955423f }, {  0.442863f,  0.238856f,  0.864188f },
{  0.162460f,  0.262866f,  0.951056f }, { -0.681718f,  0.147621f,  0.716567f },
{ -0.809017f,  0.309017f,  0.500000f }, { -0.587785f,  0.425325f,  0.688191f },
{ -0.850651f,  0.525731f,  0.000000f }, { -0.864188f,  0.442863f,  0.238856f },
{ -0.716567f,  0.681718f,  0.147621f }, { -0.688191f,  0.587785f,  0.425325f },
{ -0.500000f,  0.809017f,  0.309017f }, { -0.238856f,  0.864188f,  0.442863f },
```

```
{ -0.425325f,  0.688191f,  0.587785f }, { -0.716567f,  0.681718f, -0.147621f },
{ -0.500000f,  0.809017f, -0.309017f }, { -0.525731f,  0.850651f,  0.000000f },
{  0.000000f,  0.850651f, -0.525731f }, { -0.238856f,  0.864188f, -0.442863f },
{  0.000000f,  0.955423f, -0.295242f }, { -0.262866f,  0.951056f, -0.162460f },
{  0.000000f,  1.000000f,  0.000000f }, {  0.000000f,  0.955423f,  0.295242f },
{ -0.262866f,  0.951056f,  0.162460f }, {  0.238856f,  0.864188f,  0.442863f },
{  0.262866f,  0.951056f,  0.162460f }, {  0.500000f,  0.809017f,  0.309017f },
{  0.238856f,  0.864188f, -0.442863f }, {  0.262866f,  0.951056f, -0.162460f },
{  0.500000f,  0.809017f, -0.309017f }, {  0.850651f,  0.525731f,  0.000000f },
{  0.716567f,  0.681718f,  0.147621f }, {  0.716567f,  0.681718f, -0.147621f },
{  0.525731f,  0.850651f,  0.000000f }, {  0.425325f,  0.688191f,  0.587785f },
{  0.864188f,  0.442863f,  0.238856f }, {  0.688191f,  0.587785f,  0.425325f },
{  0.809017f,  0.309017f,  0.500000f }, {  0.681718f,  0.147621f,  0.716567f },
{  0.587785f,  0.425325f,  0.688191f }, {  0.955423f,  0.295242f,  0.000000f },
{  1.000000f,  0.000000f,  0.000000f }, {  0.951056f,  0.162460f,  0.262866f },
{  0.850651f, -0.525731f,  0.000000f }, {  0.955423f, -0.295242f,  0.000000f },
{  0.864188f, -0.442863f,  0.238856f }, {  0.951056f, -0.162460f,  0.262866f },
{  0.809017f, -0.309017f,  0.500000f }, {  0.681718f, -0.147621f,  0.716567f },
{  0.850651f,  0.000000f,  0.525731f }, {  0.864188f,  0.442863f, -0.238856f },
{  0.809017f,  0.309017f, -0.500000f }, {  0.951056f,  0.162460f, -0.262866f },
{  0.525731f,  0.000000f, -0.850651f }, {  0.681718f,  0.147621f, -0.716567f },
{  0.681718f, -0.147621f, -0.716567f }, {  0.850651f,  0.000000f, -0.525731f },
{  0.809017f, -0.309017f, -0.500000f }, {  0.864188f, -0.442863f, -0.238856f },
{  0.951056f, -0.162460f, -0.262866f }, {  0.147621f,  0.716567f, -0.681718f },
{  0.309017f,  0.500000f, -0.809017f }, {  0.425325f,  0.688191f, -0.587785f },
{  0.442863f,  0.238856f, -0.864188f }, {  0.587785f,  0.425325f, -0.688191f },
{  0.688191f,  0.587785f, -0.425325f }, { -0.147621f,  0.716567f, -0.681718f },
{ -0.309017f,  0.500000f, -0.809017f }, {  0.000000f,  0.525731f, -0.850651f },
{ -0.525731f,  0.000000f, -0.850651f }, { -0.442863f,  0.238856f, -0.864188f },
{ -0.295242f,  0.000000f, -0.955423f }, { -0.162460f,  0.262866f, -0.951056f },
{  0.000000f,  0.000000f, -1.000000f }, {  0.295242f,  0.000000f, -0.955423f },
{  0.162460f,  0.262866f, -0.951056f }, { -0.442863f, -0.238856f, -0.864188f },
{ -0.309017f, -0.500000f, -0.809017f }, { -0.162460f, -0.262866f, -0.951056f },
{  0.000000f, -0.850651f, -0.525731f }, { -0.147621f, -0.716567f, -0.681718f },
{  0.147621f, -0.716567f, -0.681718f }, {  0.000000f, -0.525731f, -0.850651f },
{  0.309017f, -0.500000f, -0.809017f }, {  0.442863f, -0.238856f, -0.864188f },
{  0.162460f, -0.262866f, -0.951056f }, {  0.238856f, -0.864188f, -0.442863f },
{  0.500000f, -0.809017f, -0.309017f }, {  0.425325f, -0.688191f, -0.587785f },
{  0.716567f, -0.681718f, -0.147621f }, {  0.688191f, -0.587785f, -0.425325f },
{  0.587785f, -0.425325f, -0.688191f }, {  0.000000f, -0.955423f, -0.295242f },
{  0.000000f, -1.000000f,  0.000000f }, {  0.262866f, -0.951056f, -0.162460f },
{  0.000000f, -0.850651f,  0.525731f }, {  0.000000f, -0.955423f,  0.295242f },
{  0.238856f, -0.864188f,  0.442863f }, {  0.262866f, -0.951056f,  0.162460f },
{  0.500000f, -0.809017f,  0.309017f }, {  0.716567f, -0.681718f,  0.147621f },
{  0.525731f, -0.850651f,  0.000000f }, { -0.238856f, -0.864188f, -0.442863f },
{ -0.500000f, -0.809017f, -0.309017f }, { -0.262866f, -0.951056f, -0.162460f },
{ -0.850651f, -0.525731f,  0.000000f }, { -0.716567f, -0.681718f, -0.147621f },
{ -0.716567f, -0.681718f,  0.147621f }, { -0.525731f, -0.850651f,  0.000000f },
{ -0.500000f, -0.809017f,  0.309017f }, { -0.238856f, -0.864188f,  0.442863f },
{ -0.262866f, -0.951056f,  0.162460f }, { -0.864188f, -0.442863f,  0.238856f },
{ -0.809017f, -0.309017f,  0.500000f }, { -0.688191f, -0.587785f,  0.425325f },
{ -0.681718f, -0.147621f,  0.716567f }, { -0.442863f, -0.238856f,  0.864188f },
{ -0.587785f, -0.425325f,  0.688191f }, { -0.309017f, -0.500000f,  0.809017f },
{ -0.147621f, -0.716567f,  0.681718f }, { -0.425325f, -0.688191f,  0.587785f },
{ -0.162460f, -0.262866f,  0.951056f }, {  0.442863f, -0.238856f,  0.864188f },
{  0.162460f, -0.262866f,  0.951056f }, {  0.309017f, -0.500000f,  0.809017f },
{  0.147621f, -0.716567f,  0.681718f }, {  0.000000f, -0.525731f,  0.850651f },
{  0.425325f, -0.688191f,  0.587785f }, {  0.587785f, -0.425325f,  0.688191f },
{  0.688191f, -0.587785f,  0.425325f }, { -0.955423f,  0.295242f,  0.000000f },
{ -0.951056f,  0.162460f,  0.262866f }, { -1.000000f,  0.000000f,  0.000000f },
```

```
{ -0.850651f,  0.000000f,  0.525731f }, { -0.955423f, -0.295242f,  0.000000f },
{ -0.951056f, -0.162460f,  0.262866f }, { -0.864188f,  0.442863f, -0.238856f },
{ -0.951056f,  0.162460f, -0.262866f }, { -0.809017f,  0.309017f, -0.500000f },
{ -0.864188f, -0.442863f, -0.238856f }, { -0.951056f, -0.162460f, -0.262866f },
{ -0.809017f, -0.309017f, -0.500000f }, { -0.681718f,  0.147621f, -0.716567f },
{ -0.681718f, -0.147621f, -0.716567f }, { -0.850651f,  0.000000f, -0.525731f },
{ -0.688191f,  0.587785f, -0.425325f }, { -0.587785f,  0.425325f, -0.688191f },
{ -0.425325f,  0.688191f, -0.587785f }, { -0.425325f, -0.688191f, -0.587785f },
{ -0.587785f, -0.425325f, -0.688191f }, { -0.688191f, -0.587785f, -0.425325f }
```

بنفس الطريقة نحسب إحداثيات القمة الحالية:

```
v_curr[0] = pframe1->scale[0] * pvert1->v[0] + pframe1->translate[0];
v_next[0] = pframe2->scale[0] * pvert2->v[0] + pframe2->translate[0];
v[0] = v_curr[0] + interp * (v_next[0] - v_curr[0]);
```

المتغير v سيحمل القيمة المحسوبة بين المشهدين للإحداثية.

قلنا أن رسم الحركة يعتمد على الزمن, سنحتاج إلى تطوير دالة نتحكم من خلالها في سرعة عرض الحركة أو مت يسمى FPS(frame per second), الدالة Animate تقوم بذلك, تأخذ أربع بارمترات, الأول و الثاني هما start و end و يمثلان المشاهد التي سنعرضها إبتداءا من start إلى غاية end و هذا طبعا سيشكل حركة.
البارامتر الثاني هو مؤشر من نوع int واسمه frame و سنعين فيه المشهد الحالي الذي سنعرض المشاهد التي بينه و بين المشهد الموالي له, قيمته نغيرها حسب تقدم الزمن.

البارامتر الرابع يمثل الزمن أو سرعة عرضنا للمشاهد, لنلقي نظرة على جسم الدالة:

```
//...
void CMD2Model::Animate(int start, int end, int *frame, float *interp)
{
  //لا ينبغي أن نتجاوز حدود المشهد الأول أو الأخير
  if ((*frame < start) || (*frame > end))
    *frame = start;
  // إذا كانت قيمة المتغير الذي يمثل الزمن تساوي 1 فإننا ننتقل إلى المشهد التالي و
0 نعطي لمتغير الزمن القيمة
  if (*interp >= 1.0f)
    {
      *interp = 0.0f;
      (*frame)++; //ننتقل إلى المشهد التالي
      if (*frame >= end) //لن نتجاوز المشهد الأخير
      *frame = start;
    }
}
//...
```

هنا نكون قد أنهينا بناء الكلاس CMD2Model, لنرى الآن كيف نستعملها.

نعود إلى الملف الرئيسي main.cpp, كنا قد أنشئنا في الدرس السابق عالم ثلاثي الأبعاد مع إمكانية التجول فيه, أعد كتابة نفس الكود, باختصار هذا هو كود الدرس السابق مع بعض التغيرات:

```
#include <stdlib.h>
#include <windows.h>
#include <GL\glut.h>
#include <stdio.h>
#include <math.h>
#include <stdio.h>
#include "MD2Model.h"
#include "textures.h"
////////////
#define PI_OVER_180  0.0174532925f
float Xrotate=0,
      Zmove=0,
      Xmove=0,
      Ymove=0,
      YAngel=0;
bool     skey[255];
unsigned int       texture[5];
float LightAmbient[]= { 1.0f, 1.0f, 1.0f, 1.0f };
float LightDiffuse[]= { 1.0f, 1.0f, 1.0f, 1.0f };
float LightPosition[]= { 0.0f, 10.0f, 10.0f, 0.0f };

void Reshape(int w, int h)
{
      glViewport(0, 0, w, h);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluPerspective(45.0, (float)w/(float)h, 1.0, 400.0);
      gluLookAt(0,3,0.1,0,3,0,0,1,0);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
}

void Display(void)
{
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
      glLoadIdentity();
      glRotatef(Xrotate,0,1,0);
      if(Xmove < -42)   Xmove = -42;
      if(Xmove > 42)    Xmove = 42;
      if(Zmove < -92)   Zmove = -92;
      if(Zmove > 92)    Zmove = 92;
      glTranslatef(Xmove, Ymove, Zmove);
      glCallList(1);
      glutSwapBuffers();
      glutPostRedisplay();
}

void Key(unsigned char key, int x, int y )
{
      if(key ==  27 )   exit(0);
}

void SpecialKeys(int key, int x, int y)
{
 skey[key] = true;
}
```

```
void action(void)
{
 if(skey[GLUT_KEY_UP])
 {
       Zmove+=(float)cos(-Xrotate*PI_OVER_180)*0.2;
       Xmove+=(float)sin(-Xrotate*PI_OVER_180)*0.2;
       YAngel+=10;
       Ymove=(float)sin(YAngel * PI_OVER_180)/5;
 }
 if(skey[GLUT_KEY_DOWN])
 {
       Zmove-=(float)cos(-Xrotate*PI_OVER_180)*0.2;
       Xmove-=(float)sin(-Xrotate*PI_OVER_180)*0.2;
       YAngel+=10;
       Ymove=(float)sin(YAngel * PI_OVER_180)/5;
 }
 if(skey[GLUT_KEY_LEFT])
 {
       Xrotate--;
 }
 if(skey[GLUT_KEY_RIGHT])
 {
       Xrotate++;
 }
}

void SpecialKeysUP(int key, int x, int y)
{
      skey[key] = false;
}

void SetupWorldList()
{
 glNewList(1, GL_COMPILE);
      glBindTexture (GL_TEXTURE_2D, texture[0]);
      glBegin(GL_QUADS);
            glNormal3f( 0.0f, 1.0f, 0.0f);
            glTexCoord2f(0.0f, 0.0f); glVertex3f(-50.0f, 0.0f,  -100.0f);
            glTexCoord2f(100.0f, 0.0f); glVertex3f( 50.0f, 0.0f,  -100.0f);
            glTexCoord2f(100.0f, 100.0f); glVertex3f( 50.0f,  0.0f,  100.0f);
            glTexCoord2f(0.0f, 100.0f); glVertex3f(-50.0f,  0.0f,  100.0f);
      glEnd();
      glBindTexture (GL_TEXTURE_2D, texture[1]);
      //Draw the border wall : back
      glBegin(GL_QUADS);
            glNormal3f( 0.0f, 0.0f, -1.0f);
            glTexCoord2f(0.0f, 0.0f); glVertex3f(-50.0f, 0.0f, 100.0f);
            glTexCoord2f(50.0f, 0.0f); glVertex3f( 50.0f, 0.0f, 100.0f);
            glTexCoord2f(50.0f, 2.5f); glVertex3f( 50.0f,  5.0f, 100.0f);
            glTexCoord2f(0.0f, 2.5f); glVertex3f(-50.0f,  5.0f, 100.0f);
      glEnd();
      //Draw the border wall : front
      glBegin(GL_QUADS);
            glNormal3f( 0.0f, 0.0f, 1.0f);
            glTexCoord2f(0.0f, 0.0f); glVertex3f(-50.0f, 0.0f, -100.0f);
            glTexCoord2f(50.0f, 0.0f); glVertex3f( 50.0f, 0.0f, -100.0f);
            glTexCoord2f(50.0f, 2.5f); glVertex3f( 50.0f,  5.0f, -100.0f);
            glTexCoord2f(0.0f, 2.5f); glVertex3f(-50.0f,  5.0f, -100.0f);
      glEnd();
```

```
            glBegin(GL_QUADS);  //Draw the border wall : left
            glNormal3f( -1.0f, 0.0f, 0.0f);
            glTexCoord2f(0.0f, 0.0f); glVertex3f(-50.0f, 0.0f, 100.0f);
            glTexCoord2f(100.0f, 0.0f); glVertex3f(-50.0f, 0.0f, -100.0f);
            glTexCoord2f(100.0f, 2.5f); glVertex3f(-50.0f,  5.0f, -100.0f);
            glTexCoord2f(0.0f, 2.5f); glVertex3f(-50.0f,  5.0f, 100.0f);
      glEnd();
      //Draw the border wall : right
      glBegin(GL_QUADS);
            glNormal3f( 1.0f, 0.0f, 0.0f);
            glTexCoord2f(0.0f, 0.0f); glVertex3f(50.0f, 0.0f, 100.0f);
            glTexCoord2f(100.0f, 0.0f); glVertex3f(50.0f, 0.0f, -100.0f);
            glTexCoord2f(100.0f, 2.5f); glVertex3f(50.0f,  5.0f, -100.0f);
            glTexCoord2f(0.0f, 2.5f); glVertex3f(50.0f,  5.0f, 100.0f);
      glEnd();
glEndList();
}

bool init(void)
{
      texture[0] = LoadTexture(".\\data\\grass.bmp");
      texture[1] = LoadTexture(".\\data\\wall.bmp");
      glEnable(GL_TEXTURE_2D);
      glClearColor(0.0f, 0.0f, 0.0f, 0.5f);
      glClearDepth(1.0f);
      glEnable(GL_DEPTH_TEST);
      glShadeModel(GL_SMOOTH);
      glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
      glLightfv(GL_LIGHT1, GL_AMBIENT, LightAmbient);
      glLightfv(GL_LIGHT1, GL_DIFFUSE, LightDiffuse);
      glLightfv(GL_LIGHT1, GL_POSITION,LightPosition);
      glEnable(GL_LIGHTING);
      glEnable(GL_LIGHT1);
      SetupWorldList();
      glBlendFunc(GL_SRC_ALPHA,GL_ONE);
      return TRUE;
}


void main(int argc, char **argv)
{
      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
      glutCreateWindow("Lesson5");
      if(!init())
      {
            MessageBox(NULL, "Initializing Error", "Error", 0);
            exit(1);
      }
      glutDisplayFunc(Display);
      glutReshapeFunc(Reshape);
      glutKeyboardFunc(Key);
      glutSpecialFunc(SpecialKeys);
      glutSpecialUpFunc(SpecialKeysUP);
      glutIdleFunc(action);
      ShowCursor(0);
      glutFullScreen();
      glutMainLoop();
}
```

قمنا باستخدام - من الدرس السابق - إكساء الأرض و الجدران, لذلك ما سنحصل عليه بعد التنفيذ هو:



كل الحركات التي يقوم بها أي موديل من موديلات لعبة Quake 2 معرفة سابقا, إلا إذا أردت إنشاء موديل بنفسك, حينها أنت تعلم من أين تبدأ و أين تنتهي حركات موديلك, بالنسبة للعبة Quake 2, نضيف مصفوفة تحمل بداية ونهاية كل حركة :

```
#include <stdlib.h>
#include <windows.h>
#include <GL\glut.h>
#include <stdio.h>
#include <math.h>
#include <stdio.h>
#include "MD2Model.h"
#include "textures.h"
//...
int animlist[21][2] =
{
    //المشهد الأول, المشهد الأخير
    {   0,  39},    // STAND
    {  40,  45},    // RUN
    {  46,  53},    // ATTACK
    {  54,  57},    // PAIN_A
    {  58,  61},    // PAIN_B
    {  62,  65},    // PAIN_C
    {  66,  71},    // JUMP
    {  72,  83},    // FLIP
    {  84,  94},    // SALUTE
    {  95, 111,},    // FALLBACK
    { 112, 122},    // WAVE
    { 123, 134},    // POINT
    { 135, 153},    // CROUCH_STAND
    { 154, 159},    // CROUCH_WALK
    { 160, 168},    // CROUCH_ATTACK
    { 196, 172},    // CROUCH_PAIN
    { 173, 177},    // CROUCH_DEATH
    { 178, 183},    // DEATH_FALLBACK
    { 184, 189},    // DEATH_FALLFORWARD
    { 190, 197},    // DEATH_FALLBACKSLOW
```

```
    { 198, 198},    // BOOM
};
```

تحدد تلك المصفوفة من أين تبدأ و أين تنتهي مشاهد كل حركة, مثلا الجري يبدأ من المشهد 40 إلى غاية المشهد 45 و هكذا, نضيف الآن enum لنعطي أسماء للحركات :

```
//...
typedef enum {
    STAND,
    RUN,
    ATTACK,
    PAIN_A,
    PAIN_B,
    PAIN_C,
    JUMP,
    FLIP,
    SALUTE,
    FALLBACK,
    WAVE,
    POINT_,
    CROUCH_STAND,
    CROUCH_WALK,
    CROUCH_ATTACK,
    CROUCH_PAIN,
    CROUCH_DEATH,
    DEATH_FALLBACK,
    DEATH_FALLFORWARD,
    DEATH_FALLBACKSLOW,
    BOOM
} AnimType;
//...
```

لا جديد حتى الآن, فهذه كلها أساسيات لغة السي ++, نصرح الآن عن متغير من نوع AnimType و آخر من نوع float يحدد الزاوية التي سندير بها الموديل على المحور y عندما نتجه إلى اليمين أو اليسار:

```
//...
AnimType animation;
float AnimationAngle = 180;
//...
```

سنستعمل بعض موديلات MD2 والتي يمكنك إما تحميلها مع المشروع المرفق مع الدرس أو تحميلها من النت أو قم بتثبيت لعبة Quake 2 كما فعلت أنا و قم بأخذ بعض الموديلات منها و يكون اسمها على الشكل name.md2 أما الإكساء المرفق مع الشخصية فيسمى مثلا name.pcx, يمكنك فتح و تغيير ملفات pcxباستعمال الفوتوشوب أو أي برنامج لمعالجة الصور, سنستعمل هنا ثلاث موديلات + ثلاث إكساءات لشخصيات اللعبة مع أحد أسلحة اللعبة, أسماء الموديلات مع الإكساءات هي:

```
tris_male.md2 + cipher.pcx
tris_female.md2 + athena.pcx
tris_cyborg.md2 + oni911.pcx
w_machinegun.md2 + Weapon.pcx
```

الموديلات محفوظة داخل المجلد models للتنظيم, نصرح الآن عن متغيرين من نوع CMD2Model أحدهما للشخصية الرئيسية و الآخر للسلاح:

```
//...
CMD2Model hero, weapon;
//...
```

نذهب إلى الدالة init() لتهيئة المتغيرين و تحميل الموديلات :

```
//...
bool init(void)
{
	//...
	hero.ReadMD2Model(".\\models\\tris_cyborg.md2"); //نحمل الشخصية الأساسية
	hero.ReadTexture(".\\models\\oni911.pcx"); //نحمل الإكساء
	weapon.ReadMD2Model(".\\models\\w_machinegun.md2"); //نحمل موديل السلاح
	weapon.ReadTexture(".\\models\\Weapon.pcx"); //و الإكساء لنا
	animation = STAND; //الحركة التلقائية هي الوقوف فقط
	//...
}
//...
```

قمنا بتحميل سخصية واحدة وهي tris_cyborg.md2, ماذا عن الشخصيات الأخرى؟ سنغير الشخصية الحالية أثناء الضغط على أخد الأزرار 1 أو 2 أو 3 من لوحة المفاتيح, نقوم بهذا على مستوى الدالة key التي تعالج أحداث لوحة المفاتيح:

```
//...
void Key(unsigned char key, int x, int y )
{
	if(key ==  27 )   exit(0);
	if(key == 49) //1 الشخصية
	{
		hero.FreeModel();
		hero.AllocateModel();
		hero.ReadMD2Model(".\\models\\tris_male.md2");
		hero.ReadTexture(".\\models\\cipher.pcx");
	}
	else if(key == 50) //2 الشخصية
	{
		hero.FreeModel();
		hero.AllocateModel();
		hero.ReadMD2Model(".\\models\\tris_female.md2");
		hero.ReadTexture(".\\models\\athena.pcx");
	}
	else if(key == 51) //3 الشخصية
	{
		hero.FreeModel();
		hero.AllocateModel();
		hero.ReadMD2Model(".\\models\\tris_cyborg.md2");
		hero.ReadTexture(".\\models\\oni911.pcx");
	}
}
//...
```

ذلك مثال عن تحميل الموديلات أثناء اللعب, و كمثال عن تغيير الحركة الحالية للموديل سنضيف حركة بسيطة و هي أن ينحني الموديل عند الضغط على الحرف c من لوحة المفاتيح, طبعا على مستوى الدالة key():

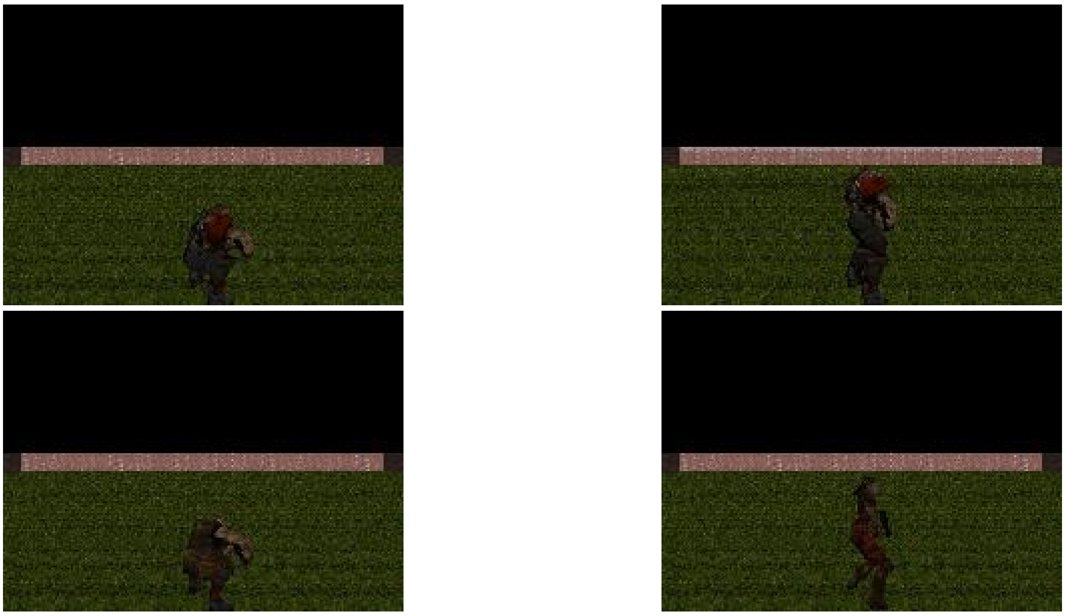
```
//...
void Key(unsigned char key, int x, int y )
{
	if(key ==  27 )   exit(0);
	if((key == 'c') || (key == 'C'))
	{
		if(animation != CROUCH_STAND)
		{
			if(animation == STAND)
				animation = CROUCH_STAND;
			else if(animation == RUN)
				animation = CROUCH_WALK;
		}
		else
			animation = STAND;
	}
	//...
}
//...
```

لنرسم الموديل والحركة الحالية له, على مستوى الدالة Display():

```
//...
void Display(void)
{
	//...
	glCallList(1);
	//////////////////////
	static int n = 0; //المشهد الحالي
	static float interp = 0.0; //تقدم الزمن
	static double curent_time = 0; //الوقت الحالي
	static double last_time = 0; //الوقت السابق
	last_time = curent_time; // نحدث الوقت السابق قبل إعطاء قيمة جديدة لمتغير الوقت الحالي
	curent_time = (double)glutGet (GLUT_ELAPSED_TIME) / 1000.0; //حساب الوقت الحالي
	interp += 10 * (curent_time - last_time); //تحديث المتغير الذي يحدد سرعة عرض المشاهد
	int start_frame, end_frame; //المشهد الأول و الأخير
	switch(animation){
		case STAND: //الوقوف
			start_frame = 0;
			end_frame = 39;
			break;
		case RUN: //الجري
			start_frame = 40;
			end_frame = 45;
			break;
		case CROUCH_STAND: //الإنحناء
			start_frame = 135;
			end_frame = 153;
			break;
		case CROUCH_WALK: //الإنحناء أثناء المشي
			start_frame = 154;
```

```
                        end_frame = 159;
                        break;
                default:
                        start_frame = 0;
                        end_frame = 39;
                        break;
        }
        glLoadIdentity(); //الموديل لن يتحرك كالجدران و الأرضية
        glTranslatef(0, 1.5, -7); //نقرب و نبعد الموديل حسب الحاجة
        glScalef(0.05f, 0.05f, 0.05f); //نصغر حجم الموديل لأنه يكون كبير جدا
        glRotatef (-90.0f, 1.0, 0.0, 0.0); //ندور الموديل لأنه يكون مقلوب
        glRotatef (-90.0f, 0.0, 0.0, 1.0);
        glRotatef (AnimationAngle, 0.0, 0.0, 1.0); // ندور الموديل حول المحور واي حسب الزاوية المحددة و التي نغيرها كلما ضغطنا على المفتاح الأيمن أو الأيسر من لوحة المفاتيح
        hero.Animate(start_frame, end_frame, &n, &interp); // نستدعي الدالة التي تحدد أي مشهد سنرسم حسب الزمن
        weapon.Animate(start_frame, end_frame, &n, &interp);
        hero.RenderFrame(n, interp); //نرسم الشخصية
        weapon.RenderFrame(n, interp); //نرسم السلاح
        //////////////////////
        glutSwapBuffers();
        glutPostRedisplay();
}
//...
```

لحساب الوقت قمنا باستعمال الدالة glutGet() مع البارامتر GLUT_ELAPSED_TIME والتي تعيد الوقت المنقضي منذ استدعاء الدالة glutInit إلى الآن عوض استخدام Timer, نفذ البرنامج وقم بتغيير الشخصيات بالضغط على الزر 1 أو 2 أو 3 أو 4 أو قم بالضغط على الزر c لتنحني الشخصية:



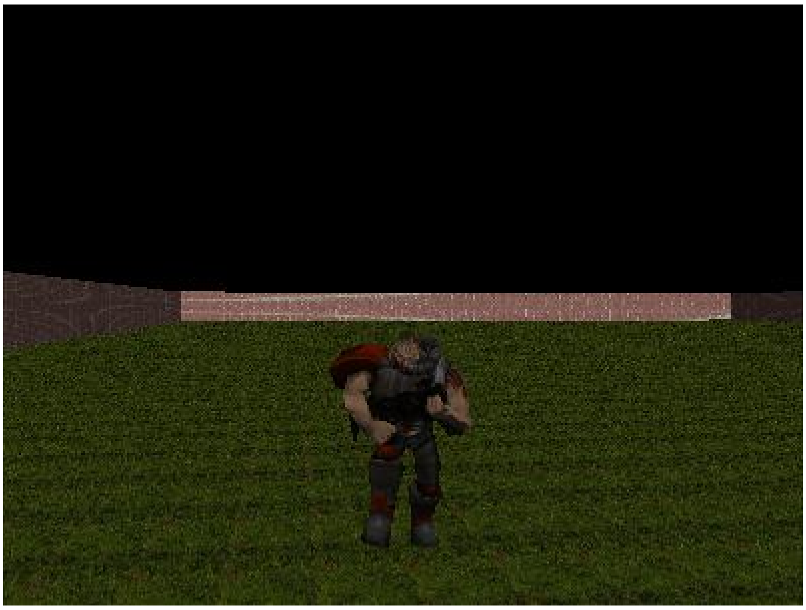
اللمسة الأخيرة التي سنضيفها في هذا الدرس هو تحريك الموديل, نريد أن تركض الشخصية المحملة إلى الأمام عند الضغط على الاتجاه الأعلى في لوحة المفاتيح و هكذا...

```
//...
void action(void)
{
 if(skey[GLUT_KEY_UP])
 {
       Zmove+=(float)cos(-Xrotate*PI_OVER_180)*0.2;
       Xmove+=(float)sin(-Xrotate*PI_OVER_180)*0.2;
       YAngel+=10;
       Ymove=(float)sin(YAngel * PI_OVER_180)/5;
       if(animation == STAND)
            animation = RUN;
       else if(animation == CROUCH_WALK)
             animation = CROUCH_WALK;
       else if(animation == CROUCH_STAND)
             animation = CROUCH_WALK;
       AnimationAngle = 180;
 }
 if(skey[GLUT_KEY_DOWN])
 {
       Zmove-=(float)cos(-Xrotate*PI_OVER_180)*0.2;
       Xmove-=(float)sin(-Xrotate*PI_OVER_180)*0.2;
       YAngel+=10;
       Ymove=(float)sin(YAngel * PI_OVER_180)/5;
       if(animation == STAND)
            animation = RUN;
       else if(animation == CROUCH_WALK)
             animation = CROUCH_WALK;
       else if(animation == CROUCH_STAND)
             animation = CROUCH_WALK;
       AnimationAngle = 0;
 }
 if(skey[GLUT_KEY_LEFT])
 {
       Xrotate--;
       if(animation == STAND)
            animation = RUN;
       else if(animation == CROUCH_WALK)
             animation = CROUCH_WALK;
       else if(animation == CROUCH_STAND)
             animation = CROUCH_WALK;
       AnimationAngle = -90;
 }
 if(skey[GLUT_KEY_RIGHT])
 {
       Xrotate++;
       if(animation == STAND)
            animation = RUN;
       else if(animation == CROUCH_WALK)
             animation = CROUCH_WALK;
       else if(animation == CROUCH_STAND)
             animation = CROUCH_WALK;
       AnimationAngle = 90;
 }
}
//...
```

الكود السابق أبسط ما يكون, وهذه بعض النتائج النهائية:







ملفات MD2 هي من أبسط الموديلات ثلاثية الأبعاد ومع ذلك أخذت منا هذا الوقت و الجهد, في دروس قادمة إن شاء الله سنرى أنواع أخرى و لعل أهم ما نريد الوصول إليه هو إدارة موديلات الـ 3D MAX المشهورة ذات اللاحقة 3ds و لكنها معقدة قليلا بالنسبة للـ MD2.

شكرا لـ David Henry لأنني استفدت كثيرا من مقاله حول موديلات MD2.

لتحميل المشروع:

http://www.mediafire.com/?tewtzznyumt

بهذا نختتم هذا الدرس وإلى درس قادم بإذن الله أترككم في رعاية الله.